Lab Assignment 2 - Lab 6

Problem Statement

Implement Backpropagation algorithm to train an ANN of configuration 2X2X1 to achieve XOR function. (Use sigmoid and Tanh activation function). You have to implement and online as well as batch gradient descent.

Theory

Backpropagation Algorithm

The backpropagation algorithm is a method used to train neural networks by adjusting their weights to minimize the error between the network's predictions and the actual outputs. It works by measuring the error at the output layer and then propagating this error backward through the network to adjust the weights accordingly.

Steps:

1. Initalize Weights and Biases Randomly
    a. Small random values for all weights (e.g., input to hidden, hidden to output).
    b. Biases often initialized to zero or small random values.
2. Forward Propagation
    a. Compute outputs layer-by-layer.
    b. For input x:
        i. Hidden Layer:
            1. $z_1 = x \times w_1 + b1$ → Linear Transformation
            2. $a_1 = activation(z_1)$ → Activation Function
        ii. Output Layer
            1. $z_2 = a_1 \times w_2 + b2$
            2. $a_2 = activation(z_2)$
    c. a_2 is the final predicted output
3. Compute Error
    a. $Error = Actual\ Output - Predicted\ Output$
4. Backward Propagation: Here we adjust weights by how much they contributed to the error.
    a. Output Layer Gradient
        i. Compute gradient of the loss w.r.t. output
            1. $dz_2 = Error \times activation\_derivative(a2)$
        ii. Compute change needed for weights and biases
            1. $dw_2 = a_1^T \times dz_2$
            2. $db_2 = dz_2$
    b. Hidden Layer Gradient
        i. Backpropagate the error to hidden layer
            1. $dz_1 = (dz_2 \times w_2^T) \times activation\_derivative(a_1)$

        ii. Compute update for input-to-hidden weights
            1. $dw_1 = x^T \times dz_1$
            2. $db_1 = dz_1$
5. Update Weights and Biases
    a. Updates using gradient descent rules
        i. $w = w + learning\_rate \times dw$
        ii. $b = b + learnint\_rate \times db$
6. Repeat for many epochs

Types of Training

1. Online Gradient Descent: Also known as Stochastic Gradient Descent, Update weights after every single input example.
2. Batch Gradient Descent: Calculate updates after seeing the entire dataset once.

Program

```
import numpy as np


#XOR DATASET

X = np.array([[0,0], [0,1], [1,0], [1,1]])

Y = np.array([[0], [1], [1], [0]])


#ACTIVATION FUNCTION

def sigmoid(x):

   return 1 / (1 + np.exp(-x))


def sigmoid_derivative(x):

   return x * (1 - x)


def tanh(x):

   return np.tanh(x)


def tanh_derivative(x):
```

```
    return 1 - np.tanh(x)**2

#ALGORITHM
def train_xor(X, Y, activation="sigmoid", epochs=10000, learning_rate=0.1, batch_mode=False):
    #Initialize weights and biases
    np.random.seed(42)
    w1 = np.random.rand(2, 2) #INPUT TO HIDDEN
    b1 = np.zeros((1, 2))
    w2 = np.random.rand(2, 1) #HIDDEN TO OUTPUT
    b2 = np.zeros((1, 1))

    #Choose activation
    if activation == "sigmoid":
        activation_function = sigmoid
        activation_derivative = sigmoid_derivative
    elif activation == "tanh":
        activation_function = tanh
        activation_derivative = tanh_derivative
    else:
        raise ValueError("Invalid activation function")

    #Train
    for epoch in range(epochs):
        if batch_mode:
            #forward pass
            z1 = np.dot(X, w1) + b1
            a1 = activation_function(z1)
            z2 = np.dot(a1, w2) + b2
```

```
    a2 = activation_function(z2)

    #backpropagation
    error = Y - a2
    dz2 = error * activation_derivative(a2)
    dw2 = np.dot(a1.T, dz2)
    db2 = np.sum(dz2, axis=0, keepdims=True)

    dz1 = np.dot(dz2, w2.T) * activation_derivative(a1)
    dw1 = np.dot(X.T, dz1)
    db1 = np.sum(dz1, axis=0)

    #updates
    w1 += learning_rate * dw1
    b1 += learning_rate * db1
    w2 += learning_rate * dw2
    b2 += learning_rate * db2
else:
    for i in range(len(X)):
        x = X[i:i+1]
        y = Y[i:i+1]

        #forward pass
        z1 = np.dot(x, w1) + b1
        a1 = activation_function(z1)
        z2 = np.dot(a1, w2) + b2
        a2 = activation_function(z2)
```

```
        #backpropagation
        error = y - a2
        dz2 = error * activation_derivative(a2)
        dw2 = np.dot(a1.T, dz2)
        db2 = dz2

        dz1 = np.dot(dz2, w2.T) * activation_derivative(a1)
        dw1 = np.dot(x.T, dz1)
        db1 = dz1

        #updates
        w1 += learning_rate * dw1
        b1 += learning_rate * db1
        w2 += learning_rate * dw2
        b2 += learning_rate * db2

      if epoch % 1000 == 0:
        loss = np.mean((Y - a2) ** 2)
        print(f"Epoch {epoch}, Loss: {loss: .4f}")

  return w1, b1, w2, b2

#EVALUATION
def evaluate_xor(X, w1, b1, w2, b2, activation="sigmoid"):
  if activation == "sigmoid":
    activation_function = sigmoid
  elif activation == "tanh":
    activation_function = tanh
```

```
    a1 = activation_function(np.dot(X, w1) + b1)
    a2 = activation_function(np.dot(a1, w2) + b2)
    return a2

# MAIN
def main():
    print("Train XOR ANN (2x2x1)")
    activation = input("Choose activation function (sigmoid/tanh): ").strip().lower()
    mode = input("Choose training mode (batch/online): ").strip().lower()
    epochs = int(input("Number of training epochs [default: 10000]: ") or 10000)
    lr = float(input("Learning rate [default: 0.1]: ") or 0.1)

    batch_mode = True if mode == "batch" else False

    print("\nTraining started...\n")
    w1, b1, w2, b2 = train_xor(X, Y, activation=activation, epochs=epochs, learning_rate=lr,
batch_mode=batch_mode)

    preds = evaluate_xor(X, w1, b1, w2, b2, activation=activation)
    print("Input\tExpected\tPredicted")
    for i in range(len(X)):
        x1, x2 = X[i]
        expected = Y[i][0]
        predicted = np.round(preds[i][0])
        print(f"{x1} {x2}\t   {expected}\t\t   {predicted}")

if __name__ == "__main__":
    main()
```

Output

1. With Online Gradient Descent
   a. Tanh Activation Function

      i.

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn7.py
Train XOR ANN (2x2x1)
Choose activation function (sigmoid/tanh): tanh
Choose training mode (batch/online): online
Number of training epochs [default: 10000]:
Learning rate [default: 0.1]:

Training started...

Epoch 0, Loss:  0.2500
Epoch 1000, Loss:  0.4908
Epoch 2000, Loss:  0.4974
Epoch 3000, Loss:  0.4986
Epoch 4000, Loss:  0.4990
Epoch 5000, Loss:  0.4993
Epoch 6000, Loss:  0.4994
Epoch 7000, Loss:  0.4995
Epoch 8000, Loss:  0.4996
Epoch 9000, Loss:  0.4996
Input   Expected        Predicted
0 0        0               0.0
0 1        1               1.0
1 0        1               1.0
1 1        0               -0.0
```

   b. Sigmoid Activation Function

      i.

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn7.py
Train XOR ANN (2x2x1)
Choose activation function (sigmoid/tanh): sigmoid
Choose training mode (batch/online): online
Number of training epochs [default: 10000]:
Learning rate [default: 0.1]:

Training started...

Epoch 0, Loss:  0.2544
Epoch 1000, Loss:  0.2509
Epoch 2000, Loss:  0.2543
Epoch 3000, Loss:  0.2589
Epoch 4000, Loss:  0.2721
Epoch 5000, Loss:  0.3630
Epoch 6000, Loss:  0.4028
Epoch 7000, Loss:  0.4224
Epoch 8000, Loss:  0.4342
Epoch 9000, Loss:  0.4421
Input   Expected        Predicted
0 0        0               0.0
0 1        1               1.0
1 0        1               1.0
1 1        0               0.0
```

2. With Batch Gradient Descent
    a. Tanh Activation Function
        i.

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn7.py
Train XOR ANN (2x2x1)
Choose activation function (sigmoid/tanh): tanh
Choose training mode (batch/online): batch
Number of training epochs [default: 10000]:
Learning rate [default: 0.1]:

Training started...

Input   Expected        Predicted
0 0        0               0.0
0 1        1               1.0
1 0        1               1.0
1 1        0               0.0
```

    b. Sigmoid Activation Function
        i.

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn7.py
Train XOR ANN (2x2x1)
Choose activation function (sigmoid/tanh): sigmoid
Choose training mode (batch/online): batch
Number of training epochs [default: 10000]:
Learning rate [default: 0.1]:

Training started...

Input   Expected        Predicted
0 0        0               0.0
0 1        1               1.0
1 0        1               1.0
1 1        0               0.0
```

Lab Assignment 2 - Lab 7

Problem Statement

Implement Backpropagation algorithm to train an ANN of configuration 3X2X2X1 to achieve majority function with 3-bit data. Output of the network must be 1 when there are two or more 1's in the data. (Use sigmoid and Tanh activation function). You have to implement and online as well as batch gradient descent.

Theory

Majority Function Dataset:

X1	X2	X3	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Program:

```
import numpy as np


# Majority Function Dataset
X = np.array([
    [0, 0, 0],
    [0, 0, 1],
    [0, 1, 0],
    [0, 1, 1],
    [1, 0, 0],
```

```
    [1, 0, 1],
    [1, 1, 0],
    [1, 1, 1]
])

Y = np.array([
    [0],
    [0],
    [0],
    [1],
    [0],
    [1],
    [1],
    [1]
])

# Activation Functions
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

def tanh(x):
    return np.tanh(x)

def tanh_derivative(x):
    return 1 - np.tanh(x)**2
```

Training Function

```
def train_majority(X, Y, activation="sigmoid", epochs=10000, learning_rate=0.1,
batch_mode=False):
    np.random.seed(42)


    w1 = np.random.randn(3, 2)
    b1 = np.zeros((1, 2))


    w2 = np.random.randn(2, 2)
    b2 = np.zeros((1, 2))


    w3 = np.random.randn(2, 1)
    b3 = np.zeros((1, 1))


    if activation == "sigmoid":
        act = sigmoid
        act_derivative = sigmoid_derivative
    elif activation == "tanh":
        act = tanh
        act_derivative = tanh_derivative
    else:
        raise ValueError("Invalid activation function")


    for epoch in range(epochs):
        if batch_mode:
            z1 = np.dot(X, w1) + b1
            a1 = act(z1)
```

```
    z2 = np.dot(a1, w2) + b2
    a2 = act(z2)

    z3 = np.dot(a2, w3) + b3
    a3 = act(z3)

    error = Y - a3
    dz3 = error * act_derivative(a3)
    dw3 = np.dot(a2.T, dz3)
    db3 = np.sum(dz3, axis=0, keepdims=True)

    dz2 = np.dot(dz3, w3.T) * act_derivative(a2)
    dw2 = np.dot(a1.T, dz2)
    db2 = np.sum(dz2, axis=0, keepdims=True)

    dz1 = np.dot(dz2, w2.T) * act_derivative(a1)
    dw1 = np.dot(X.T, dz1)
    db1 = np.sum(dz1, axis=0, keepdims=True)

    w1 += learning_rate * dw1
    b1 += learning_rate * db1
    w2 += learning_rate * dw2
    b2 += learning_rate * db2
    w3 += learning_rate * dw3
    b3 += learning_rate * db3

else:
    for i in range(len(X)):
```

```
x = X[i:i+1]
y = Y[i:i+1]

z1 = np.dot(x, w1) + b1
a1 = act(z1)

z2 = np.dot(a1, w2) + b2
a2 = act(z2)

z3 = np.dot(a2, w3) + b3
a3 = act(z3)

error = y - a3
dz3 = error * act_derivative(a3)
dw3 = np.dot(a2.T, dz3)
db3 = dz3

dz2 = np.dot(dz3, w3.T) * act_derivative(a2)
dw2 = np.dot(a1.T, dz2)
db2 = dz2

dz1 = np.dot(dz2, w2.T) * act_derivative(a1)
dw1 = np.dot(x.T, dz1)
db1 = dz1

w1 += learning_rate * dw1
b1 += learning_rate * db1
w2 += learning_rate * dw2
```

```
            b2 += learning_rate * db2
            w3 += learning_rate * dw3
            b3 += learning_rate * db3

        if epoch % 1000 == 0:
            loss = np.mean((Y - a3) ** 2)
            print(f"Epoch {epoch}, Loss: {loss:.4f}")

    return w1, b1, w2, b2, w3, b3

# Evaluation Function
def evaluate_majority(X, w1, b1, w2, b2, w3, b3, activation="sigmoid"):
    if activation == "sigmoid":
        act = sigmoid
    elif activation == "tanh":
        act = tanh

    a1 = act(np.dot(X, w1) + b1)
    a2 = act(np.dot(a1, w2) + b2)
    a3 = act(np.dot(a2, w3) + b3)
    return a3

# User chooses Activation
print("Choose Activation Function:")
print("1. Sigmoid")
print("2. Tanh")
activation_choice = input("Enter 1 or 2: ").strip()
if activation_choice == "1":
```

```python
    activation_choice = "sigmoid"
elif activation_choice == "2":
    activation_choice = "tanh"
else:
    print("Invalid choice. Defaulting to Sigmoid.")
    activation_choice = "sigmoid"

# User chooses Training Mode
print("\nChoose Training Mode:")
print("1. Batch Gradient Descent")
print("2. Online Gradient Descent")
mode_choice = input("Enter 1 or 2: ").strip()
if mode_choice == "1":
    batch_mode = True
elif mode_choice == "2":
    batch_mode = False
else:
    print("Invalid choice. Defaulting to Batch mode.")
    batch_mode = True

# Training
print(f"\nTraining using {activation_choice.upper()} activation and {'BATCH' if batch_mode
else 'ONLINE'} mode...\n")
w1, b1, w2, b2, w3, b3 = train_majority(X, Y, activation=activation_choice, epochs=10000,
learning_rate=0.1, batch_mode=batch_mode)

# Evaluate
preds = evaluate_majority(X, w1, b1, w2, b2, w3, b3, activation=activation_choice)
```

```
# Show final predictions
print("\nFinal Predictions:")
print("Input\t\tExpected\tPredicted")
for i in range(len(X)):
    x1, x2, x3 = X[i]
    expected = Y[i][0]
    predicted = np.round(preds[i][0])
    print(f"{x1} {x2} {x3}\t\t {expected}\t\t {predicted}")


while True:
    user_input = input("\nEnter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: ").strip()
    if user_input.lower() == 'exit':
        print("Goodbye!")
        break
    try:
        bits = list(map(int, user_input.split()))
        if len(bits) != 3 or any(b not in (0, 1) for b in bits):
            print("Invalid input! Please enter exactly three 0 or 1 values.")
            continue
        bits_array = np.array(bits).reshape(1, -1)
        user_pred = evaluate_majority(bits_array, w1, b1, w2, b2, w3, b3, activation=activation_choice)
        user_pred_binary = np.round(user_pred[0][0])
        print(f"Predicted Output: {int(user_pred_binary)}")
    except Exception as e:
        print("Error:", e)
        continue
```

Output

1. With Online Gradient Descent
    1. Sigmoid Activation Function

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn7.py
Choose Activation Function:
1. Sigmoid
2. Tanh
Enter 1 or 2: 1

Choose Training Mode:
1. Batch Gradient Descent
2. Online Gradient Descent
Enter 1 or 2: 2

Training using SIGMOID activation and ONLINE mode...

Epoch 0, Loss: 0.2730
Epoch 1000, Loss: 0.2633
Epoch 2000, Loss: 0.4683
Epoch 3000, Loss: 0.4811
Epoch 4000, Loss: 0.4856
Epoch 5000, Loss: 0.4880
Epoch 6000, Loss: 0.4895
Epoch 7000, Loss: 0.4906
Epoch 8000, Loss: 0.4915
Epoch 9000, Loss: 0.4921

Final Predictions:
Input           Expected        Predicted
0 0 0             0               0.0
0 0 1             0               0.0
0 1 0             0               0.0
0 1 1             1               1.0
1 0 0             0               0.0
1 0 1             1               1.0
1 1 0             1               1.0
1 1 1             1               1.0

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: 1 1 0
Predicted Output: 1

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: 0 0 1
Predicted Output: 0

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: exit
Goodbye!
```

2. Tanh Activation Function

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn7.py
Choose Activation Function:
1. Sigmoid
2. Tanh
Enter 1 or 2: 2

Choose Training Mode:
1. Batch Gradient Descent
2. Online Gradient Descent
Enter 1 or 2: 2

Training using TANH activation and ONLINE mode...

Epoch 0, Loss: 0.2912
Epoch 1000, Loss: 0.4995
Epoch 2000, Loss: 0.4998
Epoch 3000, Loss: 0.4998
Epoch 4000, Loss: 0.4999
Epoch 5000, Loss: 0.4999
Epoch 6000, Loss: 0.4999
Epoch 7000, Loss: 0.4999
Epoch 8000, Loss: 0.4999
Epoch 9000, Loss: 0.4999

Final Predictions:
Input           Expected        Predicted
0 0 0             0               -0.0
0 0 1             0               -0.0
0 1 0             0               -0.0
0 1 1             1               1.0
1 0 0             0               -0.0
1 0 1             1               1.0
1 1 0             1               1.0
1 1 1             1               1.0

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: 0 0 0
Predicted Output: 0

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: 1 0 1
Predicted Output: 1

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: exit
Goodbye!
```

2. With Batch Gradient Descent
    1. Tanh Activation Function

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn7.py
Choose Activation Function:
1. Sigmoid
2. Tanh
Enter 1 or 2: 2

Choose Training Mode:
1. Batch Gradient Descent
2. Online Gradient Descent
Enter 1 or 2: 1

Training using TANH activation and BATCH mode...

Epoch 0, Loss: 1.1183
Epoch 1000, Loss: 0.0022
Epoch 2000, Loss: 0.0000
Epoch 3000, Loss: 0.0004
Epoch 4000, Loss: 0.0002
Epoch 5000, Loss: 0.0002
Epoch 6000, Loss: 0.0001
Epoch 7000, Loss: 0.0001
Epoch 8000, Loss: 0.0001
Epoch 9000, Loss: 0.0001

Final Predictions:
Input           Expected        Predicted
0 0 0             0               -0.0
0 0 1             0               -0.0
0 1 0             0               -0.0
0 1 1             1               1.0
1 0 0             0               -0.0
1 0 1             1               1.0
1 1 0             1               1.0
1 1 1             1               1.0

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: 0 1 1
Predicted Output: 1

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: 1 0 0
Predicted Output: 0

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: exit
Goodbye!
```

2. Sigmoid Activation Function

```
ACER@Ashish MINGW64 /d/msccsit/nn/Lab Assignment (main)
$ python qn7.py
Choose Activation Function:
1. Sigmoid
2. Tanh
Enter 1 or 2: 1

Choose Training Mode:
1. Batch Gradient Descent
2. Online Gradient Descent
Enter 1 or 2: 1

Training using SIGMOID activation and BATCH mode...

Epoch 0, Loss: 0.2760
Epoch 1000, Loss: 0.1971
Epoch 2000, Loss: 0.0043
Epoch 3000, Loss: 0.0015
Epoch 4000, Loss: 0.0008
Epoch 5000, Loss: 0.0006
Epoch 6000, Loss: 0.0004
Epoch 7000, Loss: 0.0004
Epoch 8000, Loss: 0.0003
Epoch 9000, Loss: 0.0002

Final Predictions:
Input           Expected        Predicted
0 0 0             0               0.0
0 0 1             0               0.0
0 1 0             0               0.0
0 1 1             1               1.0
1 0 0             0               0.0
1 0 1             1               1.0
1 1 0             1               1.0
1 1 1             1               1.0

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: 1 0 1
Predicted Output: 1

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: 1 1 1
Predicted Output: 1

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: 1 0 0
Predicted Output: 0

Enter 3 bits separated by spaces (e.g., '1 0 1') or type 'exit' to quit: exit
Goodbye!
```